Sur une classe polynomiale relationnelle pour les CSP binaires

Wafa Jguirim, Wady Naanaa, Martin Cooper

Sur une classe polynomiale relationnelle pour les CSP binaires

Wafa Jguirim[1] **Wady Naanaa**[1] **Martin Cooper**[2] *

[1]Faculté des Sciences, Université de Monastir, Monastir 5000 Tunisie
[2]IRIT, Université de Toulouse, 31062 Toulouse.
jguirimwafa06@gmail.com wady.naanaa@fsm.rnu.tn cooper@irit.fr

Résumé

Trouver une solution à un problème de satisfaction de contraintes est en général un problème NP-difficile. Ainsi, identifier une nouvelle classe polynomiale de CSP, en d'autres termes, une classe de problèmes de satisfaction de contraintes pour lesquelles il existe une méthode de reconnaissance et de résolution de complexité polynomiale, constitue un axe de recherche fondamental dans l'étude des CSPs. Dans cet article, nous présentons une nouvelle classe polynomiale relationnelle pour les CSPs binaires. Tout d'abord nous présentons un nouvel opérateur ternaire que nous appelons mjx. Nous décrivons ensuite les relations fermées par cet opérateur en proposant un algorithme d'identification de ces relations. Afin de réduire la complexité spatiale et temporelle des traitements, nous définissons une nouvelle méthode de stockage de ces relations qui permet de réduire la complexité de chemin consistance, ce dernier niveau de consistance étant suffisant pour résoudre les instances de la classe proposée.

Abstract

Finding a solution to a constraint satisfaction problem is known to be an NP-hard task. Considerable effort has been spent on identifying tractable classes of CSP. In other words, classes of constraint satisfaction problems for which there are polynomial-time recognition and resolution algorithms. In this article, we present a relational tractable class of binary CSP. Our key contribution is a new ternary operation that we named mjx. We first describe mjx-closed relations by proposing an algorithm which identifies such relations. To reduce space and temporal complexity, we define a new storing technique for these relations which reduces the complexity of establishing path consistency, the consistency level that allows solving all instances of the proposed class.

*Martin Cooper est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210) ainsi que par l'EPSRC grant EP/L021226/1.

1 Introduction

Dans la vie quotidienne, de nombreux problèmes peuvent être définis en termes de contraintes de temps, d'espace, ou plus généralement de ressources. Nous pouvons citer à titre d'exemples : les problèmes de planification et ordonnancement, ou encore les problèmes d'affectation de ressources. Ces différents problèmes sont désignés par le terme générique CSP *Constraint Satisfaction Problem*, et ont la particularité commune d'être fortement combinatoires : il faut explorer un grand nombre de combinaisons afin de trouver une solution.

Trouver une solution à un problème de satisfaction de contraintes est, en général, un problème NP-difficile. Cependant, un grand nombre de problèmes qui se posent dans la pratique ont des propriétés particulières qui leurs permettent d'être résolus efficacement. Bien que le CSP soit NP-complet, il existe des classes d'instances qui peuvent être à la fois reconnues et résolues en temps polynomial. Dans la littérature, il existe trois types de telles classes, dites polynomiales : les classes structurelles, les classes relationnelles et les classes hybrides. Les classes structurelles sont basées sur la structure du réseau de contraintes. Ces classes exploitent les propriétés topologiques qui existent fréquemment dans les problèmes réels. Par exemple l'acyclicité du réseau [10], ou les réseaux dont la largeur arborescente est bornée par une constante [11]. Les classes relationnelles sont basées sur la sémantique des contraintes, on parle alors de langage de contraintes. Pour les caractériser on utilise l'algèbre des opérations de fermeture (polymorphismes). Parmi les classes relationnelles polynomiales, on peut citer, par exemple les CSP max-fermés [13] et les CSP médiane fermés [12, 8]. Plus récemment, des classes dites hybrides ont

été proposées. Ces classes renferment les deux notions relationnelle et structurelle. On peut citer notamment la classe des instances vérifiant la Broken-Triangle Property [7] et les CSP à rang borné [14].

Dans cette perspective, nous proposons une nouvelle classe polynomiale relationnelle définie sur les relations binaires. Notre contribution consiste à définir un nouvel opérateur que nous appelons mjx. Cet opérateur est un choix très naturel parmi les opérateurs de type majorité. Des avancés récentes [3, 2] ont permis de s'approcher d'une preuve de la conjecture de Feder et Vardi que toute classe relationnelle définie par un ensemble fini de relations est soit polynomiale soit NP-complète [9]. Cependant, d'un point de vue pratique, il reste du travail à faire pour identifier les classes polynomiales qui sont utiles dans de vraies applications. Nous espérons que cet article inspirera d'autres travaux sur la recherche de classes polynomiales vraiment utiles en pratique.

Le reste du papier est organisé comme suit : la Section 2 consiste en une revue bibliographique parcourant les notions, les définitions et notations nécessaires pour la suite. Nous donnerons aussi quelques exemples de relations fermées par mjx. Nous présenterons une caractérisation alternative de telles relations qui nous permettra de donner, dans la Section 3, un algorithme optimal d'identification des relations fermées par cet opérateur. Les instances de CSP fermées par mjx étant résolues en temps polynomial par l'établissement de la 3-consistance forte (voir Section 4), cet article inclut dans la Section 5 un nouvel algorithme de chemin consistance pour les problèmes fermés par mjx. L'algorithme proposé optimise la consistance de chemin moyennant un nouveau calcul des opérations d'intersection et de composition de relations binaires. Contrairement aux anciens travaux [4], les nouveaux algorithmes de calcul de l'intersection et de la composition possèdent une complexité linéaire $O(d)$, où d est la taille du plus grand domaine. Finalement, nous donnons une conclusion sur ce travail dans la Section 6.

2 Préliminaires

Nous rappelons d'abord quelques notions de base sur les CSP et les classes polynomiales.

Définition 1 *Un problème de satisfaction de contraintes (CSP) est défini par un triplet (X, D, C), où*

- *$X = \{x_1, ..., x_n\}$ est un ensemble de n variables.*
- *D est un ensemble fini et totalement ordonné de d valeurs, $D_{x_i} \subseteq D$ (que nous noterons aussi plus simplement D_i) étant l'ensemble des valeurs possibles pour la variable x_i.*
- *$C = \{c_1, ..., c_m\}$ est un ensemble de m contraintes chacune impliquant plusieurs variables.*

Chaque contrainte c_i est un couple $(S(c_i), R(c_i))$, où $S(c_i) \subseteq X$ est la portée de c_i (l'ensemble de variables sur lesquelles elle porte) et $R(c_i) \subseteq D_{j_1} \times \ldots \times D_{j_r}$ (si $S(c_i) = \{x_{j_1}, \ldots, x_{j_r}\}$) est sa relation de compatibilité. L'arité de c_i est le nombre de variables appartenant à $S(c_i)$. En général, on distingue les contraintes binaires, dont l'arité vaut 2, des contraintes d'arité quelconque. Les CSP binaires (dont toutes les contraintes sont binaires) sont généralement considérés de façon différente des CSP dont les contraintes sont d'arité quelconque.

Nous rappelons maintenant la notion de langage de contraintes.

Définition 2 *Un langage de contrainte Γ est un ensemble arbitraire de relations, définies sur un domaine fixe $D(\Gamma)$. On dénote par CSP(Γ) l'ensemble des instances $(X, D(\Gamma), C)$ telles que, pour tout $(S, R) \in C$, on a $R \in \Gamma$.*

Un langage de contraintes fini Γ est dit polynomial s'il existe un algorithme polynomial qui permet de résoudre toutes les instances de CSP(Γ). Un langage de contraintes infini Γ est dit polynomial si chaque sous-ensemble fini de ce langage est polynomial.

Définition 3 *Etant donnée un opérateur $\phi : D^k \to D$, une relation binaire R sur D est fermée par ϕ si, pour tout $(a_1, a'_1), \ldots, (a_k, a'_k) \in D^2$, on a $(a_1, a'_1), \ldots, (a_k, a'_k) \in R \Rightarrow (\phi(a_1, \ldots, a_k), \phi(a'_1, \ldots, a'_k)) \in R$. Un langage Γ est fermé par ϕ si toute relation de Γ est fermée par ϕ.*

Dans ce qui suit nous introduisons notre nouvel opérateur appelé mjx, un opérateur ternaire qui appartient à la classe des opérateurs de type majorité. Cette classe renferme tout les opérateurs ternaires appartenant à la famille des opérateurs de quasi unanimité [12]. Rappelons que $\phi : D^k \to D$ est un opérateur de quasi-unanimité si, pour tout $a, b \in D$, on a

$$\begin{aligned}\phi(b, a, \ldots, a) = \phi(a, b, a, \ldots, a) = \ldots \\ = \phi(a, a, \ldots, b) = a\end{aligned}$$

Définition 4 *L'opérateur* mjx *est défini comme suit :*

$$\text{mjx}(a, b, c) \stackrel{\text{def}}{=} \begin{cases} a & \text{si} \quad a = b \ \vee \ a = c \\ b & \text{si} \quad a \neq b \ \wedge \ b = c \\ \max(a, b, c) & \text{sinon} \end{cases}$$

Ainsi une relation R est fermée par mjx *si et seulement si $(a, a'), (b, b'), (c, c') \in R \Rightarrow (\text{mjx}(a, b, c), \text{mjx}(a', b', c')) \in R$*

Par ailleurs, nous rappelons le théorème suivant. Ce dernier stipule que l'ensemble des CSPs définis sur un langage fermé par un opérateur de majorité forment une classe polynomiale.

Théorème 1 *[12, 3] Soit* Γ *un ensemble de relations sur un domaine fini D. Si* Γ *est fermé par un opérateur de majorité, alors la résolution de CSP* (Γ) *est en temps polynomial.*

Il a été prouvé que la fermeture par un opérateur de majorité est une condition suffisante pour montrer que la résolution d'une instance s'effectue en temps polynomial. En effet, d'une part si une relation R est fermée par un opérateur de majorité ϕ, alors n'importe quelle contrainte (S, R) peut être décomposée en des contraintes binaires et d'autre part toute instance ne comportant que des contraintes fermée par ϕ est résolue par la 3-consistance forte [12, 3].

Corollaire 1 *Soit* $I \in CSP(\Gamma)$. *Si* Γ *est fermé par* mjx *alors la résolution de I s'effectue en temps polynomial.*

Preuve Pour démontrer que mjx est un opérateur de majorité, il suffit d'observer que mjx vérifie la propriété suivante de la famille d'opérateurs de majorité :

$$\forall a, b \quad \text{mjx}(a, a, b) = \text{mjx}(a, b, a) = \text{mjx}(b, a, a) = a$$

L'opérateur mjx est un choix évident parmi tous les opérateurs de majorité puisqu'il retourne le maximum de ses arguments lorsque ces derniers sont tous différents. Une question importante est de savoir s'il existe des relations fermées par mjx qui pourraient arriver en pratique. Nous donnons quelques exemples de telles relations. La vérification que ces relations sont bien fermées par mjx suit presque directement de la caractérisation alternative des relations fermées par mjx que nous donnons dans la Section 3.

Exemple 1 *Toute relation unaire* $x \in A \subset D$ *est fermée par* mjx. *Toute relation binaire sur domaines booléens est fermée par* mjx. *Donc la classe des instances CSP dont toutes les relations sont fermées par* mjx *peut être vue comme une généralisation de 2SAT. Les CSP médiane fermés ("connected row convex") [8] est une autre généralisation de 2SAT qui est incomparable avec les CSP* mjx *fermés.*

Exemple 2 *Soit* f *une fonction monotone décroissante, c'est-à-dire* $u < v \Rightarrow f(u) \geq f(v)$. *Toute relation de la forme* $x \geq f(y)$ *est fermée par* mjx. *Il s'ensuit que les relations binaires suivantes sont toutes fermées par* mjx, *où* x, y *sont les variables et* a, b *des constantes :*

$$x + y \geq a$$
$$(x \geq a) \vee (y \geq b)$$

Les relations suivantes sont aussi fermées par mjx *:*

$$x = y + c$$
$$(x = y + c) \vee (x \geq f(y))$$

Les relations suivantes sont aussi toutes fermées par mjx.

$$(x = a) \vee (y \geq b)$$
$$(x = a) \vee (y = b)$$
$$(x = a) \vee (x \geq f(y))$$
$$(x = a) \vee (y = b) \vee (x \geq f(y))$$
$$((x = a) \wedge (y = b)) \vee (x \geq f(y))$$

Les valeurs (a, b) *pourraient représenter des valeurs par défaut. Par exemple, si* x *est le temps du début d'une action dans un problème de planification,* $x = a$ *pourraient représenter le fait que l'on n'exécute pas l'action en question.*

3 Identification de relations mjx fermées

La manière la plus intuitive de savoir si une relation est fermée par mjx consiste à tester toutes les combinaisons possibles d'éléments qui appartiennent à la relation. Il s'agit de vérifier la fermeture de chaque image de trois paires d'éléments par mjx. Cette méthode simple engendre un nombre de tests de l'ordre de d^6. La section suivante décrit une méthode plus efficace d'identification de relations fermées par mjx.

3.1 Caractérisation des relations fermée par mjx

Nous passons maintenant à des conditions nécessaires et suffisantes de fermeture par mjx. Soit R une relation binaire. Par abus de notation, nous utilisons aussi R pour sa représentation matricielle. La matrice R^* est obtenue à partir de R en supprimant les lignes et les colonnes nulles. Les lignes et colonnes nulles sont naturellement éliminées lorsque l'on établit la consistance d'arc.

Proposition 3.1 *Soit* R *une relation binaire telle que* $R = R^*$ *(c'est-a-dire sans ligne ni colonne nulle). Si* R *est fermée par* mjx *alors*

$$(a' < b') \wedge (a, a'), (a, b') \in R \Rightarrow \forall c' > b', (a, c') \in R \quad (1)$$

et

$$(a < b) \wedge (a, a'), (b, a') \in R \Rightarrow \forall c > b, (c, a') \in R \quad (2)$$

Preuve Puisque $R = R^*$, pour chaque c' il existe c tel que $(c, c') \in R$. Par application directe de la Définition 4, nous obtenons

$$(a, a'), (a, b'), (c, c') \in R \;\Rightarrow\; (a, c') \in R$$

pour tout c' tel que $c' > b' > a'$. De façon analogue, nous obtenons

$$(a, a'), (b, a'), (c, c') \in R \;\Rightarrow\; (c, a') \in R$$

pour tout c tel que $c > b > a$.

Les conditions (1) et (2) garantissent qu'il n'existe pas de zéro précédé par deux 1 au sein d'une même ligne ou d'une même colonne lorsque que la relation est mjx-fermée.

Proposition 3.2 *Si la relation binaire R est fermée par* mjx *alors*

$$\begin{aligned} &((a, a'), (b, b'), (c, c') \in R \wedge \\ &(a > b, c) \;\wedge\; (b' > a', c') \wedge \\ &(c \neq b) \wedge (c' \neq a')) \quad\Rightarrow\quad (a, b') \in R \end{aligned} \tag{3}$$

Preuve Si $(a > b, c)$, $(b' > a', c')$, $c \neq b$ et $c' \neq a'$ alors $(\text{mjx}(a, b, c), \text{mjx}(a', b', c')) = (a, b')$.

Proposition 3.3 *Soit R une relation binaire telle que $R = R^*$. R est fermée par* mjx *si et seulement si nous avons* (1)*,* (2) *et* (3)*.*

Preuve ($\Rightarrow$) Ceci découle des Propositions 3.1 et 3.2. ($\Leftarrow$) Soit une relation binaire R. Si R vérifie (1), (2) et (3) alors quelque soit $(a, b') \notin R$, nous montrerons qu'il n'existe pas $(a, a'), (b, b'), (c, c') \in R$ qui génèrent (a, b') par mjx. Supposons que $\text{mjx}(a, b, c) = a$ et $\text{mjx}(a', b', c') = b'$ et $(a, b') \notin R$. Puisque $(a, a'), (b, b') \in R$ et $(a, b') \notin R$, nous avons $a \neq b$ et $a' \neq b'$. Puisque $\text{mjx}(a, b, c) = a \neq b$, nous avons $c \neq b$ et donc deux possibilités : soit $a > b, c$ soit $a = c$. De façon similaire, puisque $\text{mjx}(a', b', c') = b' \neq a'$ nous avons $c' \neq a'$ et les deux possibilités : soit $b' > a', c'$ soit $b' = c'$. Dans le cas $(a > b, c) \wedge (b' > a', c')$, (3) implique que $(a, b') \in R$ ce qui contredit notre hypothèse. Dans le cas $(a > b, c) \wedge b' = c'$, (2) implique également que $(a, b') \in R$. De façon symétrique, dans le cas $a = c \wedge (b > a', c')$, (1) implique encore que $(a, b') \in R$. Enfin, le cas $a = c \wedge b' = c'$ est impossible car $(c, c') \in R$ et $(a, b') \notin R$. Nous concluons ainsi que R est fermée par mjx.

Proposition 3.4 *On peut stocker une relation binaire R fermée par* mjx *en espace $O(d)$.*

Preuve Ceci est une conséquence de la Proposition 3.1 : au lieu de stocker R sous la forme d'une matrice booléenne, il suffit de stocker les positions des deux premiers 1 dans chaque ligne de cette matrice.

0	0	1	0
0	0	0	1
1	0	0	1
0	1	1	1

2
3
0
1

∞
∞
3
2

FIGURE 1 – Matrice associée à la relation mjx-fermée donnnée dans l'Exemple 3 et les deux vecteurs permettant son stockage.

Exemple 3 *Soit la relation binaire définie sur $\{0, 1, \ldots, d-1\}$ par $(|x - y| = a \;\vee\; x + y > b)$, où a et b sont des constantes telles que $b \leq 2a$. Cette relation est* mjx*-fermée. La matrice associée à une telle relation pour $d = 4$, $a = 2$ et $b = 4$, ainsi que les deux vecteurs qui permettent son stockage selon la Proposition 3.4, sont montrés dans la Figure 1.*

Il est facile, mais fastidieux, de vérifier que les exemples de relations données dans les Exemples 2 et 3 vérifient les conditions (1), (2) et (3) et donc sont bien fermées par mjx. Dans le cas où le domaine de la première variable est booléenne, les conditions (2) et (3) sont toujours vraies de façon triviale (car il n'existe pas trois valeurs a, b, c distinctes dans le domaine de cette variable) et la seule condition à remplir pour être fermé par mjx est la condition (1), ce qui nous permet de donner l'exemple suivant.

Exemple 4 *La relation $(x = a) \Rightarrow (y = b \wedge y \geq c)$, où x est une variable booléenne, y une variable quelconque et a, b, c des constantes, est fermée par* mjx*.*

3.2 Vérification de la fermeture par mjx

La vérification de fermeture par mjx consiste à vérifier les trois conditions nécessaires et suffisantes de la Proposition 3.3. La méthode la plus simple consiste à vérifier que tous les triplets d'éléments (a, a'), (b, b'), (c, c') qui appartiennent à la relation ne violent pas ces conditions. Malheureusement cette méthode engendre un nombre élevé de tests. Nous montrerons dans cette section qu'il est possible de vérifier les conditions (1), (2), (3) en temps $O(d^2)$.

Dans cette section nous supposons que la relation R à tester se présente sous la forme d'une matrice booléenne : $R(a, b) = 1$ si et seulement si (a, b) appartient à la relation. Il est raisonnable de supposer que convertir une relation donnée sous une autre forme en matrice booléenne peut se faire en temps $O(d^2)$. D'abord, nous pouvons remarquer que la vérification des conditions (1) et (2) nécessite $O(d^2)$ instructions : il suffit de parcourir chaque ligne et chaque colonne une seule fois pour vérifier que le deuxième 1 (s'il existe) de cette ligne ou colonne est suivi par une suite de 1.

Par la suite, dans cette section, nous supposons que $R = R^*$ (c'est-à-dire que les lignes et colonnes nulles de R ont été supprimées) et que R satisfait les conditions (1) et (2). Une conséquence de cette dernière supposition est que chaque zéro $R(i,j) = 0$ est précédé par au plus un 1 dans la ligne i et au plus un 1 dans la colonne j.

Pour vérifier la condition (3), pour chaque (a, b') tel que $R(a, b') = 0$, il faut vérifier qu'il n'existe pas $b, c < a$ et $a', c' < b'$ tels que

$$R(a,a') = R(b,b') = R(c,c') = 1 \\ \wedge (c \neq b) \wedge (c' \neq a') \quad (4)$$

Pour ce faire nous utilisons les structures de données suivantes :

- $NL(i,j) = \sum_{k<j} R(i,k)$ = le nombre de 1 dans la ligne i de R avant la colonne j.
- $NC(i,j) = \sum_{k<i} R(k,j)$ = le nombre de 1 dans la colonne j de R avant la ligne i.
- $N(i,j) = \sum_{k<j} NC(i,k)$ = le nombre total de 1 dans la sous matrice de R composée des lignes avant la ligne i et des colonnes avant la colonne j
- $lig1(j) = \min\{i \mid R(i,j) = 1\}$ = la ligne du premier 1 dans la colonne j de R
- $col1(i) = \min\{j \mid R(i,j) = 1\}$ = la colonne du premier 1 dans la ligne i de R

On peut établir ces structures de données en temps $O(d^2)$ par application directe de leurs définitions, ci-dessus.

Si $R(a, b') = 0$, en s'appuyant sur notre supposition que R satisfait les conditions (1) et (2), nous savons qu'il y a au plus un seul 1 dans la ligne a de R avant la colonne b' et un seul 1 dans la colonne b' avant la ligne a. Il y a donc une seule valeur $a' = col1(a)$ et une seule valeur $b = lig1(b')$ telles que $R(a, a') = R(b, b') = 1$ qui pourraient aussi éventuellement satisfaire $b < a \wedge a' < b'$. Dans le cas où nous avons $b < a \wedge a' < b'$, pour compléter la vérification de la condition (4), il faut vérifier que le nombre de paires (c, c') telles que $R(c, c') = 1 \wedge (c \neq b) \wedge (c' \neq a') \wedge c < a \wedge c' < b'$ soit 0. Le nombre de telles paires (c, c') est donné par la formule suivante

$$N(a,b') - NL(b,b') - NC(a,a') + R(b,a').$$

(le nombre de 1 dans la sous matrice de R composée des lignes avant la ligne a et des colonnes avant la colonne b' moins le nombre de ces 1 qui se trouvent dans la ligne b ou la colonne a'). Nous pouvons conclure que nos structures de données permettent une vérification de la condition (4) en temps $O(1)$ pour chaque paire de valeurs (a, b'), évitant ainsi une recherche complète sur les valeurs de a', b, c et c'. Nous avons démontré la proposition suivante.

Proposition 3.5 *Vérifier qu'une relation R est fermée par* mjx *peut se faire en temps $O(d^2)$.*

On peut même dire que la complexité de $O(d^2)$ est optimale puisque, dans le pire des cas, il faut d^2 opérations pour lire une relation binaire quelconque.

4 Résolution d'instances mjx-fermées

On appelle consistance locale le fait de vérifier que certains sous ensembles de variables ne violent pas les contraintes qui lui sont liées. Cela permet de filtrer alors certaines valeurs ou tuples impossibles avec comme conséquence un coût réduit pour la résolution de l'instance. Il existe plusieurs consistances locales, offrant chacune un équilibre différent entre efficacité du filtrage et rapidité de calcul.

On sait qu'une instance dont toutes les relations sont fermées par un opérateur de majorité, tel que mjx, est résolue soit par la 3-consistance forte [12] soit par la consistance d'arc singleton [5]. Il s'avère que dans le cas des relations fermées par mjx, la consistance de chemin directionnelle et la consistance d'arc suffisent (voir Proposition 4.1). Bien qu'intéressant au niveau théorique, ce résultat ne permet pas forcément de trouver un algorithme plus rapide. Par la suite, nous nous intéresserons plus particulièrement à la 3-consistance forte comme algorithme de résolution de CSPs mjx-fermés.

Définition 5 *Un CSP (X, D, C) est chemin-consistant (directionnel) si pour tout triplet (x_i, x_j, x_k) de variables (tels que $i, j < k$ dans le cas directionnel), et pour toute paire de valeurs (v_i,v_j) $\in D_i \times D_j$, il existe une valeur v_k de D_k telle que l'instanciation partielle ($x_i = v_i$, $x_j = v_j$, $x_k = v_k$) satisfait toutes les contraintes binaires de C portant exclusivement sur x_i, x_j et x_k.*

Proposition 4.1 *Soit I une instance CSP binaire dans laquelle toutes les contraintes sont* mjx*-fermées. Il suffit d'établir simultanément la consistance de chemin directionnelle et la consistance d'arc pour déterminer si I est satisfiable et pour pouvoir trouver une solution, s'il en existe, en temps linéaire.*

Preuve Supposons que I est arc consistante et chemin consistante directionnelle. Soit $(a_1, \ldots, a_{k-1})$ une solution partielle, c'est-à-dire une affectation aux variables $(x_1, \ldots, x_{k-1})$ qui satisfait toutes les contraintes portant sur ces variables. Par consistance d'arc, nous savons qu'une telle solution partielle existe pour $k = 3$. Nous montrerons qu'il est toujours possible (pour $k = 3, \ldots, n$) d'étendre $(a_1, \ldots, a_{k-1})$ à une solution partielle $(a_1, \ldots, a_k)$. Par récurrence, il en suivra que I

est satisfiable et que l'on peut trouver une solution en temps linéaire.

Notons R_{ij} la relation de la contrainte portant sur la paire de variables (x_i, x_j). Soit $R_{ik}(a_i)$ l'ensemble des valeurs $b \in D_k$ telle que $(a_i, b) \in R_{ij}$. Grâce à la consistance d'arc, nous savons que la représentation matricielle de R_{ij} ne comporte ni ligne ni colonne nulle. On peut donc déduire de la Proposition 3.1 que $R_{ik}(a_i)$ est de la forme

$$\{p_{ik}\} \ \cup \ \{v \mid s_{ik} \leq v \leq m_k\}$$

où $m_k = \max(D_k)$, $p_{ik} < s_{ik}$, $p_{ik} \in D_k$ et $s_{ik} \in D_k \cup \{\infty\}$ (p_{ik} et s_{ik} étant respectivement les positions du premier et du deuxième 1 dans la ligne a_i de la matrice R_{ij}). Par consistance de chemin, $\forall i, j$ tels que $1 \leq i, j < k$ et $i \neq j$, on a

$$R_{ik}(a_i) \cap R_{jk}(a_j) \ \neq \ \emptyset \qquad (5)$$

Il y a deux cas :

1. $\forall i = 1, \ldots, k-1$, $m_k \in R_{ik}(a_i)$, ou
2. $\exists i$ tel que $s_{ik} = \infty$ et donc $R_{ik}(a_i) = \{p_{ik}\}$.

Dans le premier cas, on peut choisir $a_k = m_k$. Dans le deuxième cas, on peut choisir $a_k = p_{ik}$ car $p_{ik} \in R_{jk}(a_j)$ (pour tout $j = 1, \ldots, k-1$) par (5). Dans les deux cas, $(a_1, \ldots, a_k)$ est une solution partielle.

5 Optimisation de la 3-consistance forte

L'algorithme ci-dessous décrit une procédure simple pour établir la 3-consistance forte, où AC établit la consistance d'arc.

Algorithm 1 Procédure 3C_1 (X, D, C)

```
répéter
    pour tout x_i, x_j, x_k ∈ X faire
        si (i ≠ j) ∧ (i ≠ k) ∧ (j ≠ k) alors
            R_ij ← R_ij ∩ Π_ij (R_ik ⋈ R_kj ⋈ D_k)
        fin si
    fin pour
    AC(X, D, C)
jusqu'à (pas de changement)
```

La procédure 3C_1 montre l'intérêt de pouvoir effectuer rapidement les opérations d'intersection $R \cap S$ et de composition $\Pi_{x,y}$ $(R \bowtie S)$. Dans les deux sous sections suivantes nous montrons que ces deux opérations peuvent s'effectuer en temps $O(d)$ si les relations R et S sont fermées par mjx, au lieu de $O(d^2)$ et $O(d^3)$ respectivement pour des relations quelconques.

5.1 L'intersection de deux relations mjx fermées

L'opération d'intersection s'applique sur des paires de relations comme suit :

$$R \cap S = \{(u, v) \in D^2 \mid (u, v) \in R \ \wedge (u, v) \in S\}$$

Nous utiliserons par la suite l'opération $min\_2(E)$ qui retourne la deuxième plus petite valeur de l'ensemble E. Nous supposons que chaque relation R est stockée, comme évoqué dans la proposition 3.4, au moyen des deux variables suivantes :

- $col1_R(i) = \min\{j \mid R(i, j) = 1\}$ = la colonne du premier 1 dans la ligne i de R.
- $col2_R(i) = \min\_2\{j \mid R(i, j) = 1\}$ = la colonne du deuxième 1 dans la ligne i de R.

Dans le cas où le premier 1 ou le deuxième 1 n'existe pas, la variable associée, $col1_R(i)$ ou $col2_R(i)$, prend par défaut une valeur supérieure à d que nous noterons ∞. En s'appuyant sur le fait que le deuxième 1 dans une ligne est suivi par une suite de 1, calculer l'intersection $T = R \cap S$ peut se faire à partir des règles suivantes : $\forall i$,

$$col1_T(i) \ = \ \begin{cases} col1_R(i) & \text{si } col1_R(i) = col1_S(i) \\ col1_R(i) & \text{si } col1_R(i) \geq col2_S(i) \\ col1_S(i) & \text{si } col1_S(i) \geq col2_R(i) \\ \max(col2_R(i), col2_S(i)) & \text{sinon} \end{cases}$$

et

$$col2_T(i) \ = \ \begin{cases} \max(col2_R(i), col2_S(i)) & \\ \qquad \text{si } col1_R(i) = col1_S(i) & \\ col2_S(i) & \text{si } col1_S(i) \geq col2_R(i) \\ col2_R(i) & \text{si } col1_R(i) \geq col2_S(i) \\ \max(col2_R(i), col2_S(i)) + 1 & \text{sinon} \end{cases}$$

Pour simplifier la présentation, ici nous supposons que $D_i = \{1, \ldots, d\}$ avec $d + 1 = \infty$. Pour chaque valeur de i, ces calculs s'effectuent en temps constant. Donc le calcul de $R \cap S$ s'effectue en temps $O(d)$. Puisqu'il faut retourner les $2d$ valeurs $col1_T(i)$, $col2_T(i)$ qui représentent la relation T, on peut en déduire que cette complexité est optimale.

5.2 Composition de deux relations fermées par mjx

Nous focalisons dans cette section sur l'opération de la composition qui opère sur une paire de relations R et S comme suit :

$$R \circ S = \{(u, v) \in D^2 \mid \exists w \in D, (u, w) \in R \wedge (w, v) \in S\}$$

On sait que les polymorphismes sont conservés sous les opérations de projection et de jointure [6]. Donc, si R et S sont fermées par mjx, alors $T = R \circ S$ le sera aussi. Il s'ensuit que l'on peut représenter T au moyen

des valeurs $col1_T(i)$, $col2_T(i)$ (pour chaque ligne i de la représentation matricielle de T).

Par souci de simplicité de présentation, nous supposons que les matrices R et S sont toutes les deux de taille $d \times d$.

Pour faciliter le calcul nous utiliserons les structures de données suivantes :

- $m1_R(i) = \min\{j \mid \exists k \geq i, R(k,j) = 1\}$ = la première colonne j telle qu'il existe un 1 dans au moins une des lignes $k \geq i$
- $m2_R(i) = \min\_2\{j \mid \exists k \geq i, R(k,j) = 1\}$ = la deuxième colonne j telle qu'il existe un 1 dans au moins une des lignes $k \geq i$.

Dans l'algorithme de composition, donné ci-dessous, la première boucle calcule de façon incrémentale les valeurs de $m1_S(i)$ et $m2_S(i)$ pour chaque ligne i, en commençant avec la dernière ligne.

Pour calculer la ligne i de la matrice $T = R \circ S$, nous distinguons trois cas, selon que la ligne i de R a la forme (1) $(0, \ldots, 0)$, (2) $(0, \ldots, 0, 1, 0, \ldots, 0)$ ou (3) $(0, \ldots, 0, 1, 0, \ldots, 0, 1, \ldots, 1)$. Dans chaque cas, on peut déduire la forme de la ligne i de T directement de la définition de l'opération de composition.

1. Si la ligne i de R est nulle, alors la ligne i de T sera nulle aussi.
2. Si la ligne i de R ne comporte qu'un seul 1 (dans la colonne j), alors la ligne i de T sera identique à la ligne j de S.
3. Si la ligne i de R comporte au moins deux 1 (dont les deux premiers dans les colonnes j et $k > j$), alors $col1_T(i) = \min\{col1_S(j), m1_S(k)\}$ et $col2_T(i) = \min\_2(col1_S(j), col2_S(j), m1_S(k), m2_S(k))$.

L'algorithme 2 reçoit, en entrée, deux relations R et S qui sont supposées être fermées par mjx et donc stockées sous la forme des positions des deux premiers 1 dans chaque ligne ($col1_R$ et $col2_R$ pour R et $col1_S$ et $col2_S$ pour S). L'Algorithme calcule, alors, la composition selon les trois règles données ci-dessus (deuxième boucle **pour**). Au préalable, il remplit les structures de données $m1_S$ et $m2_S$ décrites ci-dessus (première boucle **pour**). Le résultat de l'algorithme, recevant en entrée R et S, sont deux vecteurs qui mémorisent, respectivement, la position du premier et du deuxième 1 dans chaque ligne de la matrice associée la relation $R \circ S$.

La complexité de notre algorithme de composition de relations fermées par mjx est $O(d)$. Puisqu'il faut retourner les $2d$ valeurs $col1_T(i)$, $col2_T(i)$, on peut en déduire que cette complexité est optimale. Cependant, l'utilisation des algorithmes optimaux d'intersection et de composition des relations fermées par mjx dans un algorithme simple de chemin consistance, tel que 3C_1, n'est pas optimale dans le pire cas. Dans la sous section suivante nous montrerons comment construire un algorithme de chemin consistance pour les instances dont toutes les relations sont fermées par mjx, de complexité temporelle maximale $O(n^3 d \log d + n^2 d^2)$ et de complexité spatiale $O(n^2 d)$.

Algorithm 2 Fonction *Composition*$(R, S) : T = R \circ S$

```
- - R et S sont données sous la forme des vecteurs col1_R, col2_R et col1_S,col2_S
- - Le résultat, T = R ∘ S, est calculée dans col1_T et col2_T
 1: m_1 ← ∞
 2: m_2 ← ∞
 3: pour i de d à 1 faire
 4:     m1_S(i) ← min(col1_S(i),m_1)
 5:     m2_S(i) ← min_2(col1_S(i),col2_S(i),m_1,m_2)
 6:     m_1 ← m1_S(i)
 7:     m_2 ← m2_S(i)
 8: fin pour
 9: pour i de 1 à d faire
10:     j ← col1_R(i)
11:     k ← col2_R(i)
12:     si (k = ∞) alors
13:         si (j = ∞) alors - - cas 1 : ligne i de R nulle
14:             col1_T(i) ← ∞
15:             col2_T(i) ← ∞
16:         sinon  - - cas 2 : ligne i de R contient un 1
17:             col1_T(i) ← col1_S(j)
18:             col2_T(i) ← col2_S(j)
19:         fin si
20:     sinon  - - cas 3 : ligne i de R comporte deux 1
21:         E ← {col1_S(j), col2_S(j), m1_S(k), m2_S(k)}
22:         col1_T(i) ← min(E)
23:         col2_T(i) ← min_2(E)
24:     fin si
25: fin pour
26: retourner (col1_T, col2_T)
```

5.3 Algorithme de 3-consistance forte

L'algorithme de composition de la Section 5.2 montre l'utilité des structures de données $m1_S$ et $m2_S$. Nous proposons de stocker ces structures de données pour chaque relation S de l'instance, au lieu de les recalculer chaque fois que l'on souhaite recalculer la composition de deux relations R et S. Pendant l'établissement de la 3-consistance forte, les valeurs $col1_S(k)$ et $col2_S(k)$ ne peuvent qu'augmenter (suite à la suppression d'un 1 dans la ligne i de la relation S). Pour mettre à jour les valeurs de $m1_S(i)$ et $m2_S(i)$ lorsque $col1_S(k)$ ou $col2_S(k)$ augmente, il suffit de reprendre la première boucle de l'algorithme de composition à partir de la ligne $i = k$ et de continuer l'exécution de cette boucle tant que $m1_S(i)$ ou $m2_S(i)$ a changé par rapport à son ancienne valeur. Ainsi le nombre total d'opérations sera de l'ordre du nombre de mises à jour

des variables $col1_S$, $col2_S$ plus le nombre des mises à jour des variables $m1_S$, $m2_S$, ce qui représente un total de $O(d^2)$ opérations par relation, car chacune de ces quatre variables est monotone et ne peut prendre que $O(d)$ valeurs.

Lorsque $col1_R(i)$ ou $col2_R(i)$ augmente, pour mettre à jour la relation $T = R \circ S$, il faut recalculer $col1_T(i)$ ou $col2_T(i)$ en exécutant l'itération i de la deuxième boucle de l'algorithme de composition.

Il faut aussi recalculer $col1_T(i)$ ou $col2_T(i)$, en exécutant l'itération i de cette même boucle, dans les deux cas suivants :

1. lorsque $col1_S(j)$ ou $col2_S(j)$ augmente où $j = col1_R(i)$,
2. lorsque $m1_S(k)$ ou $m2_S(k)$ augmente où $k = col2_R(i)$.

Dans ces deux cas, nous utilisons des structures de données dédiées pour avoir un accès rapide aux valeurs de i telles que $j = col1_R(i)$ ou $k = col2_R(i)$:

$$\begin{array}{lcl} \forall j \quad inv\_col1_R(j) & = & \{i \mid col1_R(i) = j\} \\ \forall k \quad inv\_col2_R(k) & = & \{i \mid col2_R(i) = k\} \end{array}$$

Les structures de données inv_col1_R et inv_col2_R nécessitent en tout seulement $O(d)$ espace mémoire par relation. Nous pouvons remarquer que le maintien de ces structures de données inverses peut se faire en temps $O(\log d)$ par opération d'ajout ou de retrait (ces opérations étant nécessaires lorsque $col1_R(i)$ ou $col2_R(i)$ augmente pour une valeur de i) en utilisant, par exemple, un tas [1]. Le nombre total de ses mises à jour sera $O(d^2)$ par relation, ce qui donne une complexité totale de $O(n^2d^2 \log d)$ pour le maintien des structures de données inv_col1_R et inv_col2_R.

Le nombre maximum de fois qu'une relation portant sur deux variables x_i, x_j peut être modifiée est $O(d^2)$. Après chaque modification, il faut rétablir la 3-consistance sur le triplet x_i, x_j, x_k pour chaque troisième variable x_k, ce qui entraîne une complexité totale de $O(n^3d^2 + n^2d^2 \log d)$ pour ces propagations (y compris le maintien des structures de données inv_col1_R et inv_col2_R). L'algorithme détaillé est donné dans la Section 5.4.

Pendant l'établissement de la 3-consistance forte, il faut aussi établir et maintenir la consistance d'arc. Le nombre total d'opérations pour maintenir la consistance d'arc est $O(n^2d^2)$ si on garde en mémoire, par exemple, le minimum support pour chaque affectation (x_i, v) dans le domaine D_j de chaque autre variable x_j [4]. Cette structure de données ne nécessite que $O(n^2d)$ espace mémoire. Il s'ensuit que la consistance d'arc ne fait pas augmenter la complexité (temporelle ou spatiale) maximale. Donc, dans le cas où toutes les relations sont fermées par mjx, établir la 3-consistance forte (et donc résoudre l'instance) peut se faire en complexité temporelle $O(n^2d^2(n + \log d))$ et en complexité spatiale $O(n^2d)$.

5.4 Algorithme complet de 3-consistance forte

Dans cette sous section, nous donnons en détail l'algorithme décrit dans la Section 5.3. L'Algorithme 3 de 3-consistance forte appelle les deux sous programmes MAJ_élim_vals et Propager_PC, décrits plus loin, jusqu'à la convergence. La liste L_1 est une liste de paires (i, u) telles que la valeur u est à éliminer du domaine de la variable x_i. Lorsque l'on effectue cette élimination, il faut mettre à jour toute relation R_{ij} ainsi que toute relation R_{ji}. On rappelle qu'une relation R est stockée sous la forme des structures de données $col1_R$, $col2_R$, $m1_R$ et $m2_R$, que nous noterons $col1_{ij}$, $col2_{ij}$, $m1_{ij}$ et $m2_{ij}$ lorsque $R = R_{ij}$. Les domaines D_i sont stockés explicitement mais aussi implicitement dans les relations R_{ij} puisque les lignes et les colonnes nulles correspondent respectivement aux valeurs $u \notin D_i$ et aux valeurs $v \notin D_j$.

Les sous programmes MAJ_col12 et MAJ_m12 mettent à jour respectivement les structures de données $col1$, $col2$ et $m1$, $m2$. MAJ_col12(i,j,l,c,L_1,L_2) effectue une mise à jour éventuelle de $col1_{ij}(l)$ et $col2_{ij}(l)$ lorsque (l, c) est éliminé de la relation R_{ij}. Il est utilisé pour s'assurer que (l, c) soit éliminé de R_{ij} lorsque (c, l) est éliminé de R_{ji}. L'algorithme MAJ_m12 reprend la première boucle de l'algorithme de composition de la Section 5.2.

Algorithm 3 Procédure 3-consistance_forte(X, D, C)

1: $L_1 \leftarrow \{(i, a) \mid (x_i \in X) \wedge (a \notin D_i)\}$
2: $L_2 \leftarrow \{(i, j, a) \mid (x_i, x_j \in X) \wedge (a \in D_i)\}$
3: **répéter**
4: MAJ_élim_vals(L_1, L_2)
5: Propager_PC(L_1, L_2)
6: **jusqu'à** $(L_1 = \emptyset) \wedge (L_2 = \emptyset)$

L'algorithme Propager_PC(L_1, L_2) propage les éléments de la liste L_2 : chaque élément (i, j, l) appartenant à la liste L_2 y a été ajouté suite à un changement qui est survenu concernant la ligne l de la relation R_{ij} (c'est-a-dire dans les valeurs de $col1_{ij}$, $col2_{ij}$, $m1_{ij}$ ou $m2_{ij}$). Pour chaque troisième variable $k \neq i, j$, il faut rétablir la consistance de chemin sur les deux triplets de variables : (x_i, x_j, x_k) et (x_k, x_i, x_j). Ceci se fait de façon optimisée selon la méthode indiquée dans la Section 5.3. Les mises à jour des relations R_{ik} et R_{kj}, suite au rétablissement de la consistance de chemin, peuvent entraîner des ajouts à la liste L_1 indiquant les valeurs qui peuvent être éliminées par la consistance d'arc. La liste L_1 est traitée par le sous programme MAJ_élim_vals.

Algorithm 4 Procédure MAJ_élim_vals(L_1,L_2)

répéter
  $(i, u) \leftarrow \text{pop}(L_1)$
  si ($u \in D_i$) **alors**
    $D_i \leftarrow D_i \setminus \{u\}$
    pour $j \in X \setminus \{i\}$ **faire**
      $col1_{ij}(u) \leftarrow \infty$
      $col2_{ij}(u) \leftarrow \infty$
      MAJ_m12(i,j,u,L_2)
      pour $w \in D_j$ **faire**
        MAJ_col12(j,i,w,u,L_1,L_2)
      fin pour
    fin pour
  fin si
jusqu'à ($L_1 = \emptyset$)

Algorithm 5 Procédure MAJ_col12(i,j,l,c,L_1,L_2)

si ($col1_{ij} = c$) **alors**
  $col1_{ij}(l) \leftarrow col2_{ij}(l)$
  Ajouter (i, j, l) à L_2
  si ($col1_{ij}(l) = \infty$) **alors**
    Ajouter (i, l) à L_1
  sinon
    $col2_{ij}(l) \leftarrow \min\{b \mid (b > c) \wedge (b \in D_j \cup \{\infty\})\}$
  fin si
  MAJ_m12(i,j,l,L_2)
sinon
  si ($col2_{ij}(l) = c$) **alors**
    $col2_{ij}(l) \leftarrow \min\{b \mid (b > c) \wedge (b \in D_j \cup \{\infty\})\}$
    MAJ_m12(i,j,l,L_2)
  fin si
fin si

Algorithm 6 Procédure MAJ_m12(i,j,l,L_2)

$k \leftarrow l$
si ($k = d$) **alors**
  $m_1 \leftarrow \infty$
  $m_2 \leftarrow \infty$
sinon
  $m_1 \leftarrow m1_{ij}(k + 1)$
  $m_2 \leftarrow m2_{ij}(k + 1)$
fin si
répéter
  $change \leftarrow$ FAUX
  $m1_{ij}(k) \leftarrow min(col1_{ij}(k),m_1)$
  $m2_{ij}(k) \leftarrow min\_2(col1_{ij}(k),col2_{ij}(k),m_1,m_2)$
  si au moins une de $m1_{ij}$,$m2_{ij}$ a changée **alors**
    $change \leftarrow$ VRAI
    Ajouter (i, j, k) à L_2
  fin si
  $m_1 \leftarrow m1_{ij}(k)$
  $m_2 \leftarrow m2_{ij}(k)$
  $k \leftarrow k - 1$
jusqu'à ($k = 0$) $\vee$ ($change$ = FAUX)

Le sous programme MAJ_PC(i,j,l,k,L_1,L_2) effectue des mises à jour de la ligne l de la relation R_{ik} suite au changement dans la relation R_{ij} ou la relation R_{jk}. Il s'inspire directement de l'algorithme d'intersection de la Section 5.1 et de l'algorithme de composition de la Section 5.2. Il appelle MAJ_m12 pour mettre à jour $m1_{ik}$, $m2_{ik}$ et il appelle MAJ_col12 pour mettre à jour la relation transposée R_{ki}.

Algorithm 7 Procédure MAJ_PC(i,k,l,j,L_1,L_2)

- - recalcul de la ligne l de $R_{ij} \bowtie R_{jk}$:
$c_1 \leftarrow col1_{ij}(l)$
$c_2 \leftarrow col2_{ij}(l)$
si ($k = \infty$) **alors**
  si ($j = \infty$) **alors**
    $col1 \leftarrow \infty$
    $col2 \leftarrow \infty$
  sinon
    $col1 \leftarrow col1_{jk}(c_1)$
    $col2 \leftarrow col2_{jk}(c_1)$
  fin si
sinon
  $E \leftarrow \{col1_{jk}(c_1), col2_{jk}(c_1), m1_{jk}(c_2), m2_{jk}(c_2)\}$
  $col1 \leftarrow \min(E)$;
  $col2 \leftarrow \min\_2(E)$
fin si
- - mise à jour de la ligne l de R_{ik} :
si ($col1 \geq col2_{ik}(l)$) **alors**
  $col1_{ik}(l) \leftarrow col1$
  $col2_{ik}(l) \leftarrow col2$
sinon
  si ($col1_{ik}(l) = col1$) **alors**
    $col2_{ik}(l) \leftarrow col2$
  sinon
    $col1_{ik}(l) \leftarrow col2$
    $col2_{ik}(l) \leftarrow \min\{c \mid c > col2 \wedge c \in D_k \cup \{\infty\}\}$
  fin si
fin si
si $col1_{ik}(l)$ devient ∞ **alors** Ajouter (i, l) à L_1
fin si
MAJ_m12(i,k,l,L_2)
pour tout (l, c) éliminé de R_{ik} **faire**
  MAJ_col12(k,i,c,l,L_1,L_2)
fin pour

6 Conclusion

Dans cet article, nous avons étudié une classe relationnelle polynomiale, la classe des instances CSP binaires dans lesquelles toutes les relations sont fermées par l'opérateur mjx. Cet opérateur de type majorité est une fonction ternaire qui retourne le maximum de ces arguments lorsque ceux-ci sont tous distincts. Nous avons montré, via des exemples, que certaines relations utiles sont fermée par mjx. Nous avons aussi donné

Algorithm 8 Procédure Propager_PC(L_1,L_2)

répéter
  $(i, j, l) \leftarrow \text{pop}(L_2)$
  pour tout $x_k \in X \setminus \{x_i, x_j\}$ **faire**
    - - pour la ligne l de R_{ik},
    - - $R_{ik} \leftarrow R_{ik} \cap \Pi_{ik}(R_{ij} \bowtie R_{jk}))$:
    MAJ_PC(i,k,l,j,L_1,L_2)
    pour $(h \in inv\_col_{ij}(l) \cup inv\_col2_{ij}(l))$ **faire**
      - - pour la ligne h de R_{kj},
      - - $R_{kj} \leftarrow R_{kj} \cap \Pi_{kj}(R_{ki} \bowtie R_{ij}))$:
      MAJ_PC(k,j,h,i,L_1,L_2)
    fin pour
  fin pour
jusqu'à ($L_2 = \emptyset$)

une caractérisation alternative de relations fermées par mjx ce qui nous a permis de démontrer que de telles relations peuvent être stockées en espace $O(d)$. Une autre contribution était la réalisation d'un algorithme optimal d'identification de relations fermées par mjx de complexité $O(d^2)$.

L'exploitation de structures de données pour stocker les relations fermées par mjx dépasse le gain mémoire puisqu'elle nous a permis de proposer deux nouveaux algorithmes d'une complexité linéaire $O(d)$ de calcul de l'intersection et de la composition de deux relations fermées par mjx. Par conséquent, nous avons optimisé la complexité temporelle de la 3-consistance forte qui passe de $O(n^3d^3)$ à $O(n^2d^2(n + \log d))$ pour une complexité spatiale de $O(n^2d)$. La 3-consistance forte est une méthode de résolution des CSP binaires dont toutes les relations sont fermées par mjx.

Ces résultats montrent que l'étude d'une classe relationnelle polynomiale définie à partir d'un choix judicieux de polymorphisme peut se révéler intéressante en termes d'une méthode de stockage compact et d'un algorithme de résolution efficace. Il serait intéressant dans l'avenir d'étudier d'autres classes relationnelles définies par des polymorphismes simples dans l'espoir de trouver d'autres résultats similaires, voire même d'essayer d'établir une théorie concernant les classes relationnelles qui peuvent être résolues en temps borné par un polynôme donné. Par exemple, on peut se demander s'il existe un algorithme de résolution de complexité $o(n^3d^3)$ pour CSP(Γ) pour tout langage Γ fermé par un opérateur de majorité. Une autre perspective de recherche théorique serait d'étudier le polymorphisme de type quasi-unanimité d'arité $k > 3$ qui retourne le maximum de ses arguments sauf dans le cas où au moins $k - 1$ de ces arguments sont égaux (le cas $k = 3$ correspondant au polymorphisme mjx).

Il serait intéressant de chercher des applications dans lesquelles toutes les relations sont fermées par mjx.

Références

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1) :3, 2014.

[3] Libor Barto, Marcin Kozik, and Ross Willard. In *LICS*, pages 125–134.

[4] Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2) :165–185, 2005.

[5] Hubie Chen, Víctor Dalmau, and Berit Grußien. Arc consistency and friends. *J. Log. Comput.*, 23(1) :87–108, 2013.

[6] David A. Cohen and Peter G. Jeavons. The complexity of constraint languages. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

[7] Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees : Hybrid tractability and variable elimination. *Artif. Intell.*, 174(9-10) :570–584, 2010.

[8] Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row convex constraints. *Artif. Intell.*, 109(1-2) :243–271, 1999.

[9] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction : A study through Datalog and group theory. *SIAM J. Comput.*, 28(1) :57–104, 1998.

[10] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4) :755–761, October 1985.

[11] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2) :243–282, 2000.

[12] Peter Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artif. Intell.*, 101(1-2) :251–265, 1998.

[13] Peter Jeavons and Martin Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2) :327–339, février 1995.

[14] Wady Naanaa. Unifying and extending hybrid tractable classes of csps. *J. Exp. Theor. Artif. Intell.*, 25(4) :407–424, 2013.